CSE 551 TCPCrypt

Anna Kornfeld Simpson
Qiao Zhang
Danyang Zhuo
5/27/2015
Why we need web security?

Motivating Example:
Alice makes an online purchase using credit card.

Desired Guarantees:

Confidentiality -- Mallory cannot intercept credit card info and use it to make unauthorized purchases.

Integrity -- Mallory cannot modify the transaction details.

Authentication -- Transaction is sent to the right vendor, not some other party.
Secure Transport: SSL/TLS

SSL/TLS is a user process (e.g. OpenSSL lib)

Interposes between Application and Transport Layers
modified application
unmodified TCP

Provides a secure TCP channel between two parties

Application-level protocols: PGP, SSH etc.
SSLv3/TLS handshake

**Msg1:** clientHello, supported ciphers, client nonce

choose secret $S$, compute

$$K = f(S, R_{Alice}, R_{Bob})$$

**Msg2:** serverHello, chain of certificates, chosen cipher, server nonce

compute

$$K = f(S, R_{Alice}, R_{Bob})$$

**Msg3:** secret encrypted with server's public key, keyed hash of prev msgs to ensure cipher suite not downgraded!

$\{S\}_{Bob}, \{\text{keyed hash of handshake msgs}\}$

**Msg4:** keyed hash of prev msgs to prove no-tempering of prev msgs + server has private key + server knows session keys derived from $K$

data protected with keys derived from $K$
SSLv3/TLS cannot be ubiquitous

SSL/TLS achieves

1. authentication (usually only server to client)
2. data integrity through crypto hash
3. confidentiality through symmetric encryption

Simplified SSLv3/TLS
(Kaufman’s Network Security)
SSLv3/TLS cannot be ubiquitous

However, SSL/TLS is

1. too expensive due to server-side decryption
2. hard to set up and use
   a. library not easy to use
   b. certificate is a pain point
3. not suitable for all applications
4. cannot get encryption/integrity without authentication
   e.g. if no server certificate, no encryption of data

code snippet for openSSL?
**tcpcrypt: encrypt all TCP traffic**

Symmetric key encryption is cheap  
$\Rightarrow$ we can feasibly encrypt all TCP traffic

Confidentiality/Integrity are general-purpose primitives  
Authentication is application-specific!  
$\Rightarrow$ we should decouple confidentiality/integrity with authentication

Tcpcrypt proposes a new architecture:  
Embed encryption/integrity checking into TCP as TCP extensions  
Provide hooks to enable flexible application level authentication
tcpcrypt security guarantees

By default, all TCP traffic are encrypted
   Protects against passive eavesdropping without ANY app modification
   But can be man-in-the-middled

Tcpcrypt key exchange generates a session id on both end points
   If the session ids match, then guarantee no MitM.
   Session id can be used for authentication
tcpcrypt vs SSL

High server performance: push decryption to clients

As a TCP option
  Applications use BSD socket API
  Encryption automatically enabled if both end points support tcpcrypt
  Backwards compatible, graceful fallback to vanilla TCP/SSL

New getsockopt() returns session_id as hook for authentication
  certificate-based authentication
  password-based authentication
tcpcrypt handshake

- Server sends supported public ciphers
- Client sends public key & supported symmetric ciphers
- Server generates & encrypts symmetric master key under client’s public key
- Client decrypts and now both sides have the symmetric key
Putting the Handshake in TCP

Regular TCP setup

\[\text{SYN} \rightarrow \text{SYN ACK} \rightarrow \text{ACK}\]

tcpcrypt part 1

\[\text{SYN - CRYPT(HELLO)} \rightarrow \text{probe tcpcrypt} \rightarrow \text{SYN ACK} \rightarrow \text{ACK}\]
Putting the Handshake in TCP

Regular TCP setup

- SYN
- SYN ACK
- ACK

Tcpcrypt full handshake

- SYN - CRYPT(HELLO) probe tcpcrypt
- SYN ACK - CRYPT(PKCONF) public key ciphers and key sizes list
- ACK - CRYPT(INIT1) symmetric ciphers and MACs list, nonce, public key
- ACK - CRYPT(INIT2) encrypted client and server nonce (master key)

crypto on
Only 1 extra message? How?

tcpcrypt full handshake

- First two messages encode info in TCP options
- INIT1 and INIT2 too large for options, so have to use application data
- Delays application data by one RTT (from 3rd message to below “crypto on”)

SYN - CRYPT(HELLO)
    probe tcpcrypt

SYN ACK - CRYPT(PKCONF)
    public key ciphers and key sizes list

ACK - CRYPT(INIT1)
    symmetric ciphers and MACs list, nonce, public key

ACK - CRYPT(INIT2)
    encrypted client and server nonce (master key)

crypto on
Options can have up to 40 bytes.
A tcpcrypt encrypted packet

<table>
<thead>
<tr>
<th>src port</th>
<th>dst port</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq no.</td>
<td></td>
</tr>
<tr>
<td>ack no.</td>
<td></td>
</tr>
<tr>
<td>d.off.</td>
<td>flags</td>
</tr>
<tr>
<td>window</td>
<td>checksum</td>
</tr>
<tr>
<td>options (e.g., SACK)</td>
<td>MAC option</td>
</tr>
</tbody>
</table>

MACed

Encrypted

(64-bit seq)

(64-bit ack)

TCP length
A tcpcrypt encrypted packet

- Data is encrypted (confidentiality)
- Headers + data are MAC’d (integrity)
- Also MAC dashed items (not in header)
- Don’t MAC ports, checksum so NATs still work
Session resumption - no latency!

tcpcrypt init handshake

- SYN - CRYPT(HELLO) probe tcpcrypt
- SYN ACK - CRYPT(PKCONF) public key ciphers and key sizes list
- ACK - CRYPT(INIT1) symmetric ciphers and MACs list, nonce, public key
- ACK - CRYPT(INIT2) encrypted client and server nonce (master key)

Session Resume

- SYN - NEXTK1 New session based on session with ID X
- SYN ACK - NEXTK2 OK!
- ACK crypto on
- ack
Great, how does that work?

- Session caching and session ID like in SSL
- NEXTK1 = 9 bytes of next session ID
- Fall back to full handshake if cache miss
Great, how does that work?

- Session ID calculated by HMAC’ing a value with the session secret
- Session secret built during init by HMAC’ing the initial parameters and then iterate over time
- HMAC is a keyed hashing algorithm.
Certificate-based Authentication

- Server signs the shared Session ID with its private key
- Batch Sign
  - SIGN requires encryption using server’s private key which is as expensive as standard SSL
  - Server can sign a batch of Session IDs to amortize of the cost of SIGN function
  - Session ID is not secret
Weak Password Authentication

- Client and server share a secret
- **C→S**
  - MAC(hash(salt, realm, secret), TAG_C || Session ID)
- **S→C**
  - MAC(hash(salt, realm, secret), TAG_S || Session ID)
- Client and server can verify if the other end knows hash(salt, realm, secret) or not
Strong Password Authentication

- In Weak Password Authentication, adversary can impersonate the server to get the \( h = \text{hash}(\text{salt, realm, secret}) \) and then use a dictionary to guess the secret
- Use Diffie-Hellman problem to generate a shared key
  - \( h_0 = H_0(\text{password, user name, server name}) \)
  - \( h_1 = H_1(\text{password, user name, server name}) \)
  - \( g \) is a generator of group \( G \) (order of \( G \) is a prime number \( q \))
  - \( U, V \) are randomly chosen in \( G \)
Strong Password Authentication

Server:
g, h0, g^{h1}, U, V, beta

Client:
g, h0, h1, U, V, alpha
Strong Password Authentication

Server:
g, h0, g^{h1}, U, V, beta

Client:
g, h0, h1, U, V, alpha

g^{(alpha)} * U^{(h0)}
g^{(beta)} * V^{(h0)}
Strong Password Authentication

Server:
g, h0, g^h1, U, V, beta
g^(alpha) * U^(h0)

Client:
g, h0, h1, U, V, alpha
g^(beta) * V^(h0)
Strong Password Authentication

Server: g, h0, $g^{h1}$, U, V, $\beta$
g$^\alpha$

Client: g, h0, h1, U, V, alpha
g$^\beta$
Strong Password Authentication

**Server:**
g, h0, \(g^{h1}\), U, V, beta
\(g^{(alpha)}\)
\(g^{(alpha*beta)}\)
\(g^{(beta*h1)}\)

**Client:**
g, h0, h1, U, V, alpha
\(g^{(beta)}\)
\(g^{(alpha*beta)}\)
\(g^{(beta*h1)}\)
Strong Password Authentication

Server:
g, h0, g^h1, U, V, beta

Client:
g, h0, h1, U, V, alpha
g^(alpha)
g^(beta)

h = H(h0, g^(alpha), g^(beta), g^(alpha*beta), g(beta*h1))

C->S : MAC(h, TAG_C || Session ID)
S->C : MAC(h, TAG_S || Session ID)
Implementation

- **Linux Kernel Implementation**
  - Port OpenSSL into kernel for RSA support
  - Incompatible with TCP segment offloading

- **Userspace Implementation**
  - Use divert socket to access TCP packets
  - Track connections, calculate checksum, rewrite sequence number,...
  - IPC call for getsockopt
Implementation

- OpenSSL
  - Modify OpenSSL’s BIO layer to leverage the shared Session ID
  - Use a single worker thread to batch sign all incoming SSL connections
Connection Rate

Main bottleneck is public key operations. RSA decryption is much slower than encryption.
Authentication Cost

Authentication can be 25x faster with batching.
Latency

tcpcrypt has lower latency than SSL.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Connect time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.2</td>
</tr>
<tr>
<td>TCP cached</td>
<td>0.3</td>
</tr>
<tr>
<td>TCP not cached</td>
<td>11.3</td>
</tr>
<tr>
<td>TCP cmac</td>
<td>11.4</td>
</tr>
<tr>
<td>TCP PAKE</td>
<td>15.2</td>
</tr>
<tr>
<td>SSL cached</td>
<td>0.7</td>
</tr>
<tr>
<td>SSL not cached</td>
<td>11.6</td>
</tr>
<tr>
<td>TCP</td>
<td>105</td>
</tr>
<tr>
<td>TCP cached</td>
<td>103</td>
</tr>
<tr>
<td>TCP not cached</td>
<td>219</td>
</tr>
<tr>
<td>TCP cmac</td>
<td>320</td>
</tr>
<tr>
<td>TCP PAKE</td>
<td>426</td>
</tr>
<tr>
<td>SSL cached</td>
<td>210</td>
</tr>
<tr>
<td>SSL not cached</td>
<td>321</td>
</tr>
</tbody>
</table>
Demo: Wireshark + TCP options

sudo ./launch_tcpcryptd.sh  

http://tcpcrypt.org/fame.php

wireshark (passive eavesdropping)  
ip.addr == 171.66.3.196

general fallback

without “sudo ./launch_tcpcryptd.sh”

cleartext GET request
Q: Why don’t people actually deploy tcpcrypt? Is it because people don’t care about security or people are happy with SSL’s performance?

ewm87: “people get very nervous about changing security protocols”

eamullen: “Maybe if it the implementation was verified, people would be more ready to adopt it.”

jtoman: “Sure you could implement these primitives using getsockopt, but good luck doing that in PHP running on apache (or even PHP running via FCGI).”

bornholt: “It's one of those clean sheet designs that wasn't viable when current systems were being designed but that makes perfect sense now (like, say, Rust).”
Q: Why don’t people actually deploy tcpcrypt? Is it because people don’t care about security or people are happy with SSL’s performance?

vsriniv2:
1. Deploying TCP options is impossible on today's internet.
2. tcpcrypt is susceptible to attacks that dramatically restrict its utility -- downgrade and MITM attacks are possible
3. When Http is treated as a transport (albeit a not-great one, with pipelining but not split transactions), encryption and authentication are better deployed at that layer.

billzorn:
“I just don't think people care enough about security to catalyze a major shift in something as massive and boring as infrastructure. I don't see how encrypting all of my traffic would help that much when there are still major security concerns about things like browsers and OSs.”
Q: Should we have encryption in transport layer in the first place?

lijl: “I still think encryption should be done in the application level. This is definitely more a religious question, but adding encryption to the transport layer just seems to break the abstraction provided by the OSI model. And to the ubiquitous problem, is it possible to simply add a layer in the OS network stack or even in glibc to encrypt all network messages?”

antoine: “Besides the religious reasons mentioned above, I would also like to mention a more practical point: updates.”
Q: Do you agree that it makes sense to separate confidentiality/integrity from authentication? Are the first two guarantees more fundamental than the third one? Do you ever only need the first two guarantees without the third one?

wysem: “it seems to make sense to build confidentiality/integrity into the network stack itself (at TCP) instead of forcing this on application developers. It adds a level of protection for everyone and allows those requiring more to add at the application level.”

“Given the increasingly mobile nature of computing, are mobile processors implementations of hardware encryption/AES sufficiently efficient to promote use of this idea? “

naveenks: This should help me remain anonymous and be certain that my traffic can't be snooped by anyone or be tampered by anyone.
Q: Does it make sense to further separate confidentiality from integrity and push one of them further down the network stack?

wysem: “It seems to make sense to build confidentiality/integrity into the network stack itself (at TCP) instead of forcing this on application developers. It adds a level of protection for everyone and allows those requiring more to add at the application level.”

“Given the increasingly mobile nature of computing, are mobile processors implementations of hardware encryption/AES sufficiently efficient to promote use of this idea?”

naveenks: “I can imagine cases where I want just integrity but not confidentiality.”
Comments:

jrw12: “This paper's analysis of what's wrong with modern crypto on the web is right on.”

naveenk: “how easy would it be to build a Tor like system on top of tcpcrypt?”